Integrating UEFI services with ATF and u-boot FIT on mbed Linux

Bryan O’Donoghue
Linaro member services
bryan.odonoghue@linaro.org
Introduction

- Convergence in cloud management of edge devices
- Secure management services
  - Amazon
    - [https://aws.amazon.com/iot-device-management/](https://aws.amazon.com/iot-device-management/)
  - Arm Pelion
    - Arm and Intel are collaborating on Pelion
  - IBM
  - Microsoft Azure
- What about secure boot?
  - Fragmented
  - Sometimes vendor-specific
- Secure update?
  - Cloud interface well defined
  - Low-level device side?
    - Not so much
Secure boot on Arm - the Trusted Board Boot Requirements

- **BL1 (BootROM)**
  - Hardware specific boot-agent baked into silicon
  - References some kind of One-Time-Programmable (OTP) key
  - Uses this key to verify 1st stage bootloader (BL2)
- **BL2 (verified by BL1)**
  - Contains a key which it uses to validate Firmware Image Package (FIP)
  - A FIP is basically an archive with boot components in it
    - Grub
    - OP-TEE
    - DTB
  - FIP contents can be signed and verified as part of a Root-of-Trust (RoT)
- **BL3 (Verified by BL2)**
  - u-boot/grub/other
- ATF doesn't mandate any specific subsequent steps
  - [https://chromium.googlesource.com/external/github.com/ARM-software/arm-trusted-firmware/+v1.4-rc0/docs/trusted-board-boot.md](https://chromium.googlesource.com/external/github.com/ARM-software/arm-trusted-firmware/+v1.4-rc0/docs/trusted-board-boot.md)
UEFI secure boot

- EDKII or Tianocore
  - Conceptually similar to ATF/FIP - your laptop is EFI based
- Differences in terms of key ownership and management
  - On x86 it is generally the case you can take ownership of your EFI keys
  - On Arm in the embedded space it is not expected you can take ownership of keys
- Can be used to sign
  - Kernel
  - InitramFS
  - Bootscripts
- UEFI has a set of well defined key types with specific functions like the TBBR
  - Platform key (PK)
    - Vendor key - Dell, Lenovo, Microsoft
    - Used to validate the KEK
  - Key exchange key (KEK)
    - Used to update and validate DB
  - DataBase (DB)
    - Set of allowed signatures
  - Database eXcluded (DBX)
    - Set of excluded signatures
u-boot FIT Images

- u-boot has a mechanism to SHA/RSA images inside an archive called a FIT image
- The u-boot key is baked into the image
- It is the responsibility of the system designer to authenticate u-boot
- U-boot FIT authentication begins and ends with u-boot
- Allows you to verify
  - Kernel
  - Initrd
  - Bootscript
u-boot UEFI

- UEFI has
  - BootTime services
  - RunTime services

- u-boot implements some BootTime services
  - EFI memory map
  - EFI printing for early kernel
  - EFI application booting
    - Allows you to launch any UEFI application
    - Including a kernel with the UEFI boot stub

- u-boot RunTime services
  - Mostly/entirely not supported
  - UEFI has a callback to ExitBootServices() at which point u-boot implements few to no RunTime services
  - But, it could

- Why u-boot?
  - Board vendors already have u-boot in their boot flow.
  - Doesn’t require replacing ATF with Tianocore
  - Gives the option to have BootROM boot a verified u-boot in a vendor solution
Is UEFI secure boot more secure than u-boot FIT?

- Not that I can tell
  - Open to correction on this point
- Done right the ability to revoke or add new keys is powerful
- How the keys are stored is non-trivial
  - Is it the exclusive domain of SecureWorld?
  - Is vendor differentiation legitimate within the standard?
- My feeling is
  - Pros
    - In theory a drop-in replacement for u-boot FIT
    - Brings benefits of key addition/revocation
  - Cons
    - Requires real development work to get going
    - What if your system doesn’t support or hasn’t implemented a TrustZone firmware.
So why choose UEFI Secure Boot if it not more secure than u-boot FIT?

- A well exercised and well supported Secure Boot method
  - Thanks to x86
- DB and DBX
  - Powerful and definitely interesting for secure systems in the field
- Capsule Update
  - UEFI provides a RunTime Service called UpdateCapsule()
  - Load a capsule into memory via kernel provided character device
  - Reboot system to update
  - Runtime OS need not know how to update firmware
    - NAND, NOR, eMMC, whatever, Runtime OS doesn’t need to know this
    - So the Linux side kernel and user mode code doesn’t need to be platform specific
  - Capsule update is signed
- Compatible with ATF
  - Your existing ATF bootstrap, FIP signing stays as is
  - You update it with RunTime or BootTime UEFI services provided by u-boot, OP-TEE or a combination of the two
Mbed Linux before and after

- **ATF/FIP**
  - BL1
  - BL2
  - FIP image
    - OP-TEE
    - u-boot

- **u-boot**
  - FIT with signed components
    - Kernel
    - Initrd
    - Bootscript

- **Linux**
  - Booted via standard “bootm” or “bootz” command
  - Board-specific update

- **ATF/FIP**
  - BL1
  - BL2
  - FIP image
    - OP-TEE
    - u-boot

- **u-boot**
  - UEFI SecureBoot Signed
    - Kernel
    - Initrd
    - Bootscript

- **Linux**
  - Booted via efi bootstub “bootefi” command
  - Capsule update method means no board-specific update logic required in Linux
Questions and references

- https://www.pelion.com/
- https://www.linuxjournal.com/content/take-control-your-pc-uefi-secure-boot
- https://os.mbed.com/docs/mbed-linux-os/v0.8/welcome/index.html
- https://github.com/siemens/efibootguard
- https://www.rodsbooks.com/efi-bootloaders/controlling-sb.html#creatingkeys
- https://blog.hansenpartnership.com/the-meaning-of-all-the-uefi-keys/
Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org